仮想記憶を意識した大次元行列固有値解析 (数値計算のアルゴリズムの研究)

AUTHOR(S):

村田, 健郎

CITATION:

村田, 健郎. 仮想記憶を意識した大次元行列固有値解析 (数値計算のアルゴリズムの研究). 数理解析研究所講究録 1974, 199: 49-97

ISSUE DATE:

1974-01

URL:

http://hdl.handle.net/2433/107325

数理解析研究所講究録
199巻 1974年 49-97

仮想記憶を意識した大次元行列固有値解析

日立 神奈川工場 村田 健郎

目 次

仮想記憶を意識した大次元行列固有値解析

日立　神奈川工場　　村田　健郎

§1. はじめに

バーチアル（Virtual）と云う英語の意味を辞書で調べてみると，仮の，という意味と本當の，という意味の両義があることが判る。バーチアルメモリ方式の計算機の出現によって従来ネックとされていた主メモリ容量の制限から開放されて見ると，今度は処理時間ネックが表面化する。その理由の第一は，次元数を増すとき，容量は次元数の二乗に比例するに過ぎないが演算回数は三乗に比例することが多いこと，そして誤差の累積に対処するための諸手当を要すると云う行列演算の本質から来るものである。その第二の理由は，バーチアルにメモリが増大したに過ぎないことに伴う，所謂バーチアルオーバーヘッドである。バーチアルメモリをうまく使いこなすには，局所性の高いプログラムを書けとよく云われるが，それはマトリックス計算の現場ではどう云うことかについて若干報告したい。バーチアルメモリからかりそめでなく

本當の幸福を引き出す一助になれば幸いである。

初めに要約めいた云い方をすると凡そ次のようになる。

(1) 通常あるプログラムをたゞ次元を大きくとるだけの変更でバーチアル方式の機械にかけると，勿論うまく解ける場合もあるが，解けることは解けるが，ひどいアイドルタイムを生じてバーチアル公害を起すことがある。つまり実質的にはドラムコンピュータに近い状態で走り，そのため生じたアイドルタイムは他ジョブによる回收が困難なことがある。

(2) 密行列に対する連立一次方程式関係（ガウス消去法，ガウスジョルダン法，クラウト法，コレスキー法）では，通常の教科書で慣行となっている演算順序では，FORTRANの世界で不都合が起り易い。逆にPL/1の世界ではそのまゝで良いことが多い。しかしこの場合の演算順序は単なる慣習であって，それを直すのは容易である。

(3) ブロック帯行列，或いは帯行列専用に作られた連立一次方程式関係の諸プログラムは，大局的には高能性が高くバーチアルにとつて大変好都合である。しかしブロックの一つの次元が100元を超すとか，帯の巾が200を超すと(2)に於

けると同様の注意を局部についても払う必要がある。

(4) 固有値解析の方は連立一次方程式関係のようには簡単には行き兼ねるが，やはりバーチャル向きのアルゴリズムと比較的容易にバーチャル向きに改造できるアルゴリズム，そして本質的にバーチャル向きでないアルゴリズムの三種に分類できる。固有値問題は従来のリアルメモリ方式の機械では連立一次方程式のようにはうまく外部メモリを使いこなせなかった難物であるから，これこそバーチャルによって未開の分野を開きたいものである。因みに，千元の密行列の固有値全部が「超高性能電子計算機」（大型プロジェクト機の正式名，今後㊙機と略称する。ユーザー用のリアルメモリ領域1.65 MB）でもって1時間のCPUタイムでできる。USEタイムも約4時間だから，現実的にできると云って良い。千元の密行列は8 MBだからこれは相当うれしい話である。

§2. 基本演算 A±B，AB，A⁻¹B について。 周知のように，FORTRANでは，A(I,J,K) のI,J,Kが1からNまで動くとき，Iが1増せば主メモリの番地も1増すが，Jが1増せば主メモリの番地はN増し，Kが1増せば主メモリの番地はN^2増しとなる。一方バーチャルメモリ方式の機械では，4KB程度か

ら成る「頁」と呼ばれる単位毎に，物理的には次の三つの状態の何れかをとっている。

(1) 実メモリ上ではまだ主メモリにはでていなくてドラムにある場合．

(2) 実メモリ上でも主メモリに存るが，それはバーチアル対リアルの対照対がASR（アソシアティブレジスタ）と呼れるものに登録されてない状態にある場合．

(3) 実メモリ上で主メモリに存ることは勿論，バーチアル対リアルの対照対がASRに登録されている場合．

実際には最近の大型計算機例えばHITAC8800/8700やIBMの370/168ではバーチアルメモリ方式と同時にバッファメモリ方式も採用しているので，上述の(3)が更に細分されて，

(3-1) バッファメモリには未だとりこまれてない場合

(3-2) バッファメモリに既にとりこまれている場合．

と云うように分れる。それでこう云う計算機の所謂平均命令実行時間は，次のような式で示される。

$$T_{eff} = T_0 + \beta T_2 + \tau T_1 + \delta T_d$$

(3-2)
(3-1)
(2)
(1)

この式の右辺の各項が凡そどれ位かを，行列の基本演算で

見積る所から今日の話を始めよう。

§2-1　A±B　これは簡単すぎる。

§2-2　B := AB　Bは縦に長い行列，Aは正方行列または帯行列であることが多い。X = AX と書いて見るとこれは反復法の基本演算だなと云うイメージが鮮明になる。今，AはN×Nの正方行列，BはN×Pの行列但しN≫Pだとして以下の二つのプログラムについて調べて見る。主メモリは1MB，但しユーザに与えられるリアルメモリは700KB程度としよう。

```
      DO 30 K=1,P
      DO 20 I=1,N
      ACC=0.0D+0
      DO 10 J=1,N
      ACC=ACC+A(I,J)*B(J,K)
10    CONTINUE
      C(I)=ACC
20    CONTINUE
      DO 30 I=1,N
      B(I,K)=C(I)
30    CONTINUE
```

```
      DO 20 I=1,N
      DO 5 J=1,N
      ROW(J)=A(I,J)
5     CONTINUE
      DO 20 K=1,P
      ACC=0.0D+0
      DO 10 J=1,N
      ACC=ACC+ROW(J)*B(J,K)
10    CONTINUE
      CT(K,I)=ACC
20    CONTNUE
      DO 30 K=1,P
      DO 30 I=1,N
      B(I,K)=CT(K,I)
30    CONTINUE
```

P=N/10位だと，N=50なら $T_{eff} \fallingdotseq T_0$ で左も右も実行時間に差異はない。バッファメモリ16KBだとN=100では $T_{eff} \fallingdotseq T_0+\beta T_6$ となり，左の簡単なプログラムの方では β が6%程度に対し

右の方のプログラムは2%程度，T_Bが1.5μsで左右では60nsの差となる。T_0が600nS程度の機械だと，その差は1割たからよしたことないが，下が200nsの機械だと50nsの差ができるとすこし気になって来る。 $N=250$ となると，$T_{eff} \fallingdotseq T_0+\beta T_b+\tau T_t$ となり，左のプログラムでの $\beta \fallingdotseq 10\%$，$\tau \fallingdotseq 5\%$ に対し右のプログラムでは $\beta \fallingdotseq 3\%$，$\tau \fallingdotseq 1\%$ 程度となり，T_tが3.5μsとすると左と右の差は

$$1500(0.1-0.03)+3500(0.05-0.01)=105+140=245\ ns$$

と云うことで，可なり違って来る。

$N=500$ となると $500\times500\times4=1MB$ だから今考えている700KBのリアルメモリに於てはドラム参照の方がむしろ主要部を占める。つまり $T_{eff} \fallingdotseq T_0+\beta T_b+\tau T_t+\delta T_d$ の最後の項が左のプログラムでは $\delta=0.05$ (5%)，右のプログラムでは 0.002 (0.2%) となって，$T_d=10\ ms$ として左の δT_d が500μs，右の δT_d が20μsとなる。以上1頁=4KB，ASRは16頁，バッファは16KB 1セクターは1KBの超機で単精度で計算するとし、見つもった。

§2-3 $X=A^{-1}B$ 即ち $AX=B$ の解Xを求めるプログラム ガウス消去法でやる場合の前進消去法の部分を調べることにしよう。反復法に使用するときのことを考えて，前進消去の終った段階で，Aが下三角と上三角の二つに分解されているようにしておく。簡単のためピボット選択とか ill condition

チェックは省略してある。

```
      DO 10  K=1,N-1                    DO 10  K=1,N-1
      DO 10  I=K+1,N                    DO  5  I=K+1,N
      A(I,K) = -A(I,K)/A(K,K)           A(I,K)=-A(I,K)/A(K,K)
                                      5 CONTITUE
      DO 10  J=K+1,N                    DO 10  J=K+1,N
                                        DO 10  I=K+1,N
      A(I,J) = A(I,J) +A(I,K)*A(K,J)    A(I,J) = A(I,J) +A(I,K)*A(K,J)
   10 CONTINUE                       10 CONTINUE
```

左のプログラムがバーチアルを意識しない通常の教科書通りのプログラムである。このプログラムの所要時間は最内側のループの中の演算　A(I,J) = A(I,J) + A(I,K) * A(K,J)　で主要部が決る。左のプログラムではこれが，DO 10 J=K+1,N となっているから，A(I,J)　A(K,J) 共に1回計算する毎に J が1増すのに対して主メモリ上ではこれらが何れもN番地飛び離れたところを参照する。　A(I,J) = A(I,J) + A(I,K) * A(K,J) は高々10命令で，超機だと 250nS x 10 = 2500nS = 2.5μS 程度だから，これに対しもし毎回バッファメモリの入れかえが起れば 1.5μS のオーバーヘッド，ASR の入れかえが毎回起れば 3.5μS のオーバーヘッドと云うことになってしまう。毎回ドラムの参照を起すとなったら 10×10^3 μS と云うことで破局的だ

が，そうでなくても，1000回に1回のドラム参照でも10μsと云うことで，これが主要部になる。右のプログラムは，最内側のループが．

```
DO 10   I = K+1, N
A(I,J) = A(I,J) + A(I,K) * A(K,J)
10   CONTINUE
```

となっているから，1回の演算の後，Iが1増すにともなってA(I,J)もA(I,K)も，直ぐ隣りの番地に移るだけだから具合が良いのである。このようにするために，

```
DO 5  I = K+1, N
A(I,K) = -A(I,K) / A(K,K)
5   CONTINUE
```

によってA(I,K)を先に作ってしまう必要がある。そのためアンダーラインしてあるカードを2枚余計パンチすれば良い。

§2-2のマトリックスの積のプログラムの左と右の相異も同様の主旨のものであった。即ち，左のプログラムでは，最内側ループの演算 ACC = ACC + A(I,J)*B(J,K) の A(I,J)の J が動くのが痛いので，それをさけるために，ROW(J)と云う一次元アレイを余計に用意して ROW(J) = A(I,J) とやることによって，主要部は ACC = ACC + ROW(J) * B(J,K) で済むようにしたのであった。さて話を連立一次方程式の方に

もとそう。　右のプログラムは左のものと比べると格段に優れているけれども，N^2 がリアルメモリ容量を超えると，やはりSTAが利いて来てそうべきな問題まで解けると云うわけには行かない。そこでオーソドックスには，元のマトリックスを小行列を要素とするブロックマトリックスにして，ブロックガウス消去法で解くと云うことになるわけであるが，従来あるプログラムの外ワクを変えないで，局部的なプログラムの修正で何とか乗り切る方法に，「往復法」と「次元縮約法」がある。上のプログラムに「往復法」を適用すると次のようになる。

```
      DO 15  K=1,N-1
      SIGN = 1.0
      DO 5  I = K+1, N
      A(I,K) = - A(I,K)/A(K,K)
5     CONTINUE
      SIGN = -1.0 * SIGN
      IF ( SIGN . LT. 0.0 ) GO TO 12
      DO 10  J = K+1, N
      DO 10  I = K+1, N
      A(I,J) = A(I,J) + A(I,K) * A(K,J)
10    CONTINUE
      GO TO  15
12    NI = N+K+1
      DO 13  J = K+1, N
      DO 13  I = K+1, N
      A(I, NI-J) = A(I, NI-J) + A(I,K) * A(K, NI-J)
13    CONTINUE
15    CONTINUE
```

このプログラムで，アンダーラインした所が往復法のための追加である。主メモリ容量に対して必要なバーチャルメモリ容量が高々1.5倍位までならこんな簡単な修正でもアイドルタイムは大巾に縮減できる。

以上は，既に使っているプログラムの外ワクを変えないで局所的な修正によってバーチャル向にする手法を例示した。

これに対して，初めからプログラムする場合には，プログラム全体のプロセスをよく見通したうえで，選択するアルゴリズムと DIMENSION とをよく適合させてスッキリしたプログラムを作成できることが多い。

例えば二次元アレイ $A(I,J)$ に対してコレスキー法を行う場合，$A = LL^T$ でなく $A = U^TU$ から出発すれば DIMENSION と算式が適合する。コレスキー法を一次元アレイで行う場合には，どちらの算式でも，それに合せた配列を自由に設定できる。

コレスキ法は対称行列を相手にするものだから容易なことは当然であるが，クラウト法についてはどうか？クラウト法による LU 分解は，

$(r=1,n)$

$$\sum_{k=1}^{r-1} l_{ik}u_{kr} + l_{ir} = a_{ir} \quad (i=r,n)$$

$$\sum_{k=1}^{r-1} l_{rk}u_{ki} + l_{rr}u_{ri} = a_{ri} \quad (i=r+1,n)$$

から導びかれるから，ガウス消去法のように初めに $A(I,J)=a_{JI}$ と云う風に DIMENSION を設定しただけでうまく通常の算式と適合する；と云うように簡単には行かない。方程式 $AX=B$ を B を X にとりかえて何度も何度も解くと云う所謂逆反復法は，固有値解析問題に於て特に重要だから，A が疎の場合，密の場合それぞれに対して最適のプログラムを用意しておくべきである。

§3. 実対称固有値問題とバーチアルメモリ方式

実対称固有値問題に話を限っても，密行列の場合と帯行列の場合とで話がちがうし，必要なのは固有値だけなのか，固有ベクトルも必要なのか，それもどれ位の個数必要なのか，重複或いは近接固有値があるのかないのか，ある場合どれ位の精度で分離せねばならぬのか，などなど事情の如何によって処方が大いに変ってまともに調べたら大変なことである。そこで今日は，Wilkinson－Reinsch の Linear Algebra（文献(1)）にのっていると云う意味で著名な算法に範囲をしぼり，それらがバーチアルメモリ方式にとってどうかと云うことに焦点をしぼって調べることにする。但し Jacobi 法のように明らかに大次元に向かないものは始めからとり上げない。また，Householder－Bisection 法や，Householder－QR 法の類のものについては，三重対角化の所だけがバーチアルにとってどうかが問題になる。と云うわけで，結局調べることにしたのは次の四つである。

(1) Householder's Tridiagonalization (II/2)

(2) QR Algorithm for Band Symmetric Matrices (II/7)

(3) Tridiagonalization of a Symmetric Band Matrix (II/8)

(4) Simultaneous Iteration Method (II/9)

§3-1 Householder's Tridiagonalization (其2)

リアルメモリ方式のコンピュータでは，メモリに入り切れないほど大きいマトリックスに対しては実際上不可能と云って良い。Wilkinson の本[2] の 294 頁に Householder's Process on a computer with a two-level store と題してこのような場合に対処する方法が示されてはいるが，今日の 8800/8700 クラス の機械の内部処理速度と，ディスクに対するアクセスタイムとのはなはだしい不釣合から，この方法で果して現実的な時間内で 1,000 元を超えるマトリックスの三重対角化ができるかどうか？。労力に見合った効果を期待することは殆んど不可能であろう。一方バーチャルならば，Fortran レベルでのプログラムの若干の手直しによって n^2 がリアルメモリ容量の 2～3 倍程度なら，許容可能なアイドルタイムの範囲内で計算できる。この際バーチャルだからと云って何の手当もしない普通のプログラムでやるとひどいアイドルタイムを生じてバーチャル公害を起す。ついでながら，二次元アレイに対するアルゴリズムでやるより，一次元アレイ（$\frac{1}{2}n(n+1)$長）についてのアルゴリズムでやる方が，記憶容量の節約に止まらず，局所性が生かされて断然好ましい。

以上は密行列に対するハウスホルダ法について述べたわけであるが，帯行列に対してはどうか．実はこれがうまく行く

と大変具合が良いのだが，普通のハウスホルダー法では帯巾を $2m+1$ とするとき一回の三重対角化毎に帯巾が $m-1$ だけ拡がるので，バーケアルと云えども使用範囲が極度に限定される。例えば帯巾が 200，次元が 5,000 の帯行列に於ては，頭の約 50 列を三重対角化した所で残りは約 4,950 元の密行列と化してしまう。そこで今，筆者は帯巾を 2倍或いは 1.5 倍までの拡大を許して，局所性の高い拡張ハウスホルダ法とでも云う可き手法を研究中である。これについては §4 で述べることにする。

§3-2 QR Algorithm for Band Symmetric Matrices

これは帯に対する直接QR法とでも云う可きものである。と云うわけは，QR法は密行列或いは巾の広い帯行列に対して直接やると收束が遅くてかなわぬので，通常は三重対角化を経由して三重対角行列に対して実施する。それに対して，この方法は帯の巾が適当に狹く，また仗要とされる固有値の数も適当にすくないとき，メリットを出そうとするものであるからである。帯巾は完全に元のまゝと云うわけには行かないが，片側に m だけ拡がるだけで済むから，收束の速さとの兼合いで，三重対角化を経由する普通のやり方は前節でも述べたように直ぐ密行列化すると云う事情も手傳って，メリ

ソトを主張できる分野があるわけである。さてアルゴリズムの概要は以下の通り。

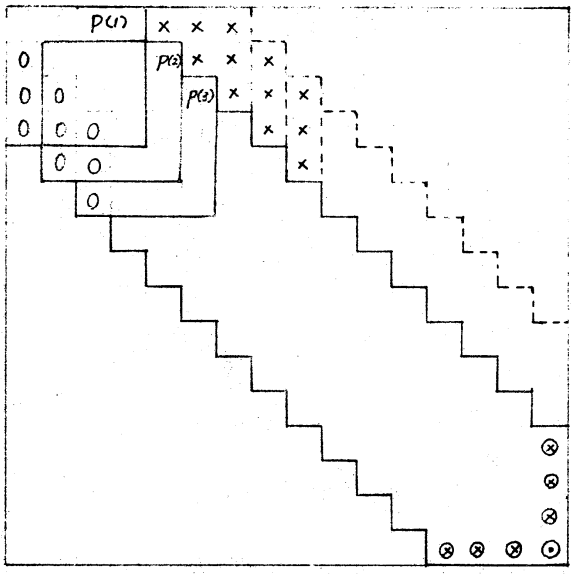
与えられた対称帯行列 A_1 に対して

(1) $A_s - k_s I = Q_s R_s$ (QR分解)

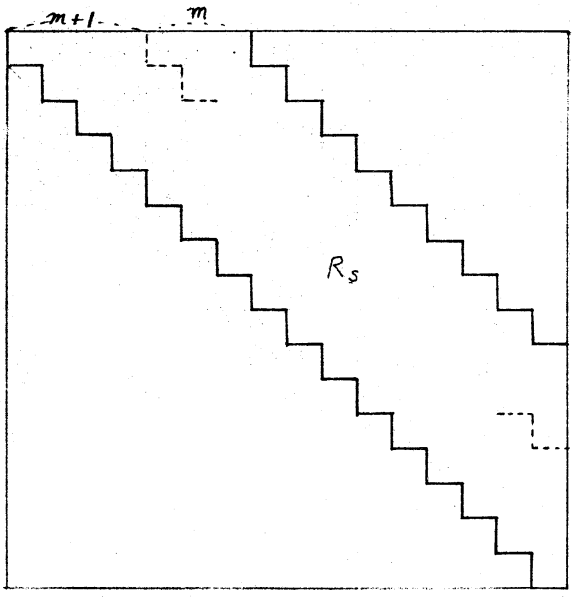
(2) $A_{s+1} = R_s Q_s$ (帯行列の積)

を繰返す。ここに k_s はシフト数でこれを毎回適当にえらぶと，図3-1の⊗印の所が急速に0に収束し，従って⊙の値が固有値の一つに急速に収束する。(1)の操作を $A_1 - k_1 I = Q_1 R_1$ についてもっと詳く調べてみると図3-1の左上段の所に示すように，elementary orthogonal tr $P_1^{(i)}$ $(i = 1, 2, \cdots)$ を $A_1 - k_1 I$ に対して順次行う所謂 Householder の三角化分解（三重対角化でないことに注意）であっ

(図3-1)

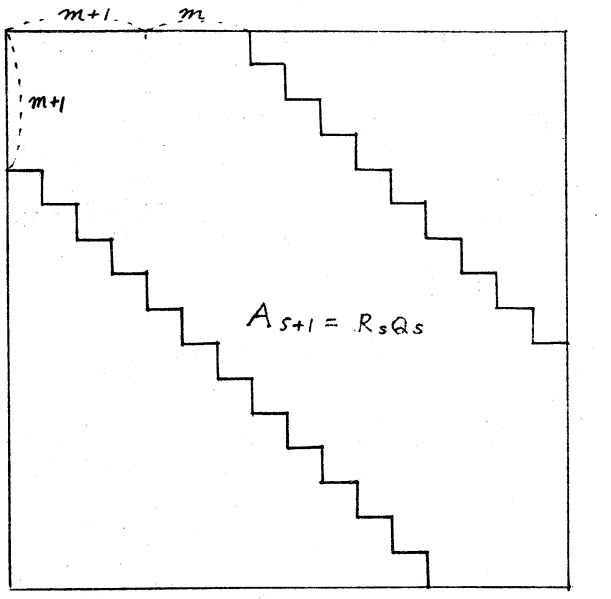


(図3-2)



て，$Q_1 = P_1^{(k)} P_1^{(k-1)} \cdots P_1^{(2)} P_1^{(1)}$ である。この結果として得られる R_1 は図3-2に示すような右上三角(?)の帯行列であり，$R_1 Q_1$ は図3-3に示すような非対称の帯行列である。こうして，$A_2 = R_1 Q_1$ ができたあと $A_2 - k_2 I$ に対して丁度前と同じく，$A_2 - k_2 I$ の左下を消去する操作を $P_2^{(i)}$ $(i = 1, 2, \cdots)$ によって行うわけであるが，もう帯巾が広がることはない。…。と云うわけで，三重対角化のように(は)帯がひろがらないから，バーチアルであるなしにかゝわらず好都合である。特にバーチアルと云う見地からは，$P_j^{(i)}$ の大きさが $m+1$ 次元(一定)である矣，局所性も良好と云える。また $A_s \to A_{s+1}$ 一段当り，帯の端から端まで参照するのは，QR分解に一回と，できたQRを逆順に掛け直すことに一回だけだから，$A_1 \to A_2 \to \cdots \to A_{ss}$ の回数が小さければ，局所性は全体としても良好になる見込みがある。次に調べる，Rutischauser - Schwarz の帯に対する三重対角化法よりも，バーチアルにとっては有望の方法と思われる。問題は収束の

(図3-3)



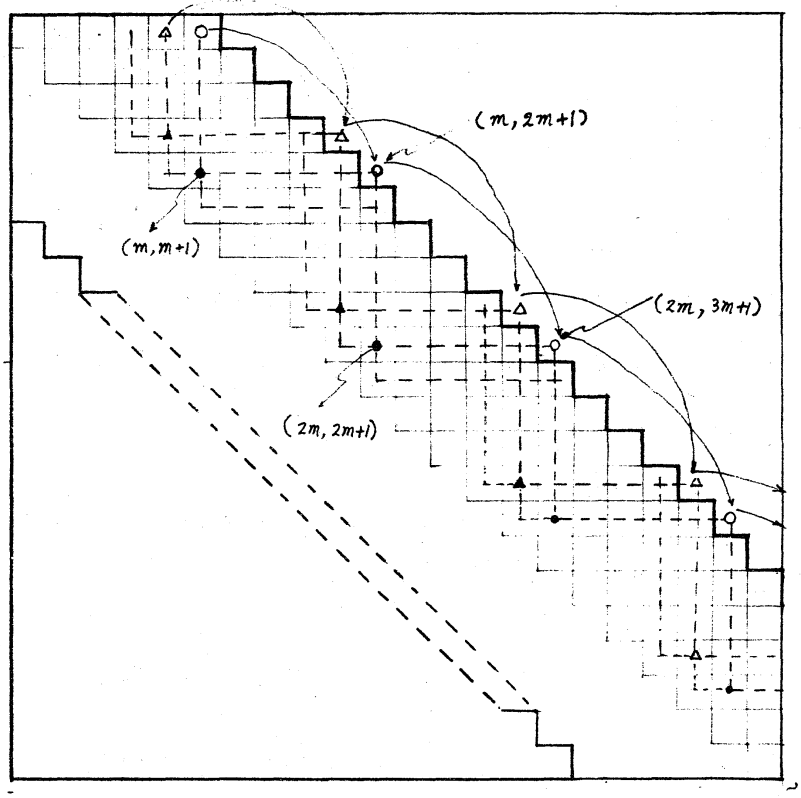
速さと云うことになるので，帯巾を予じめ後の節で述べる方法で半分とか四分の一にするとかして，收束がうんと速くなるようだと面白い。

§3-3 Tridiagonalization of a Symmetric Band Matrix (Ⅱ/8)

右の図を見ていただく。これはm＝5の帯行列であるが，やり方の概要は以下の通り。まず右の図で(1,6)の要素，即ち一般的に云えば(1,m+1)の要素を消すための plane Rotation

$U(m, m+1, \varphi_1)$

(図3-4)

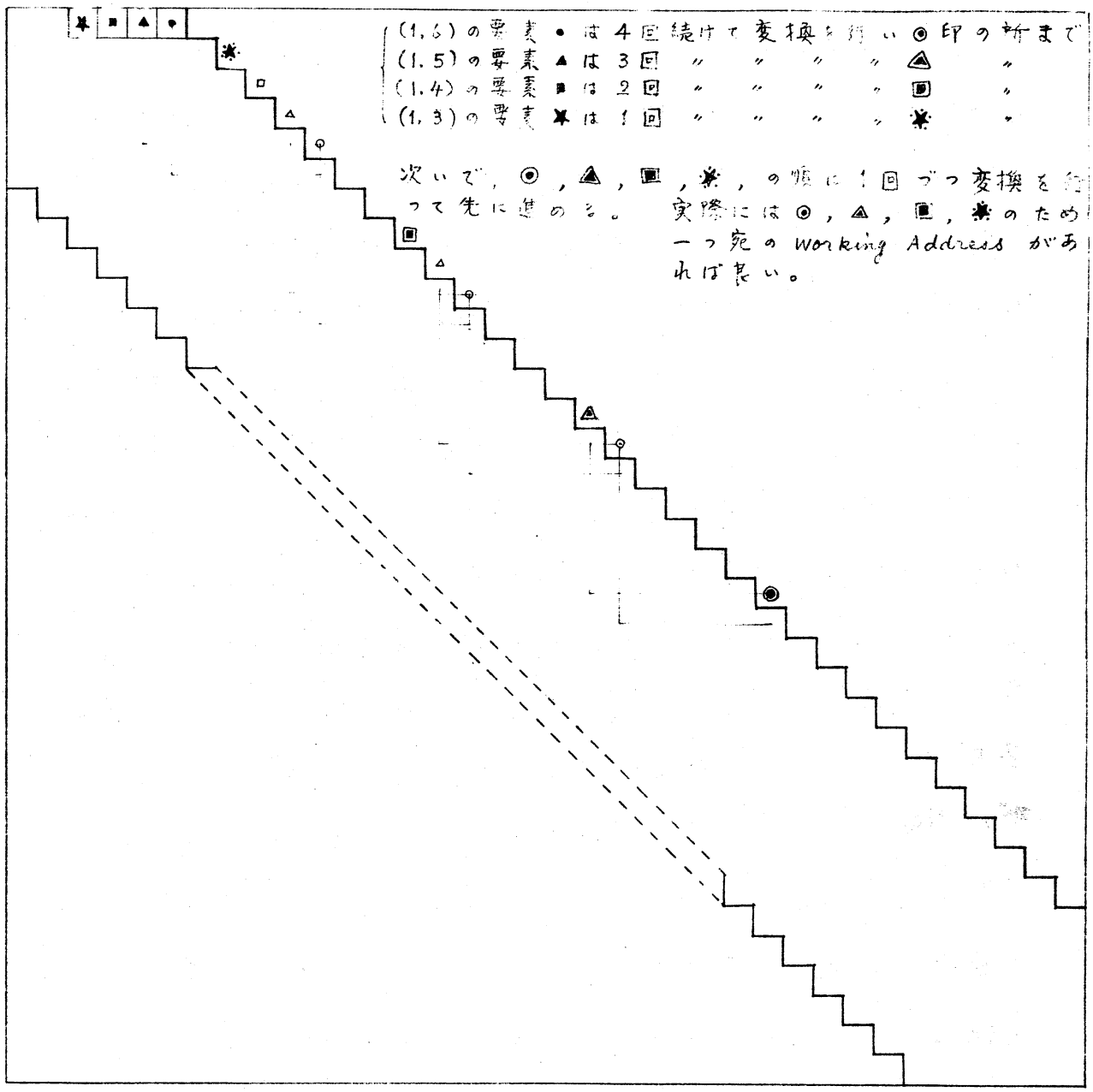


を行うと，帯の外の一矣 (m, 2m+1) に non zero ができる。それを消すため plane Rotation $(2m, 2m+1, \varphi_2)$ を行うとまた帯の外の矣 (2m, 3m+1) に nonzero ができる。……。こうして non zero がマトリックスの外に飛び出すまで追いかけてやるのである。それがすんだら今度は (1,5) 一般的に云えば (1,m) について

今と同様のことを行う。…。こうして(1,3)まで消し去れば，第一行（そして同時に第一列）が三重対角化されたことになる。以下同様にして第二行，第三行…と順に三重対角化するわけであるが，この方法で帯の長さの方は次第に短くはなって行くけれども，一つの要素毎に必ず右下端まで参照するから，局所性は非常にわるい。

そこで一つ一つの要素毎に右下端まで追いかけるのを止めて，一行分まとめてやることにする。そのためには m-1 個のワーキングアドレスを用意すれば済むことであるから，バーチアルにとってはこの方が断然好ましい。これで，1列分に対して1度だけ右下端まで参照と云うことで，可なり良くなった。これをもっと進めて，p(m-1) 個のワーキングアドレスを用意して，p行の三重対角化に1回だけ帯の右下端まで参照と云う風にプログラムすることもできる。帯だけで主メモリ容量を大巾にはみ出すような場合にはこのような工夫が要る。しかし，この方法は，SQRT が planeRotation 毎に必要であるとか，基本演算回数が他の方法と比べて多いとか欠点が多い。後で述べる「拡張ハウスホルダ法」の方が優れていると思われる。また，「部分的用直交化を行うランチョス法」の方が優れていると思われる。

図3-4-1 Wilkinson-Reinsch Linear Algebra II の改良法.



§3-4 Simultanious Iteration Method (11/9)

Subspace Iteration とも Jennings の方法とも呼ばれ，最近有限要素法の応用分野で評判のものである。密行列に対して適用すると，必要な固有値，固有ベクトルの数 p が行列の元数れに比べて可なり小さいときでも，Householder 法で三重対角化を行ってのち Bisection法－Inverse Iteration と云うオーソドックスな方法と比べて必ずしも良くない。また帯の場合，帯巾が十分狭くて，帯に対する直接QR法の収束が速いときとか，必要とする固有値の数が極度にすくなくて，所謂 Determinant 法が有利になるときにはそちらに道をゆずるけれども，比較的広巾で，比較的小数の固有値（and or）固有ベクトルを求めるとき，特に有能とされている。もーつみのがせないのは，有限要素法での動的解析で質量マトリックスが帯のとき，$A\varphi = \lambda B\varphi$ の形の固有値問題を扱うが，Householder法やQR法の系統では，Aが折角帯でも，これを標準化すると帯でなくなってしまうのに対し，帯の長所を保有したまゝで計算することができることである。（この点は Determinant 法も同様である。）さて，ここでは，文献(1)の例と，文献(3)の記述に大筋を従って，$A\varphi = \lambda\varphi$ の大きい方からp箇の固有値を求めると云う典型的な場合についてバーチァルと云う観点から調べて見た。帯行列Aによる一次変換が大次元の場合の主役だから高所作業は良い。

（アルゴリズムの大筋）

a) start with $h=0$, $k=2$

b) choose X_0 with orthonormal columns

c) $\overline{X}_1 = A X_h$,

perform $k-1$ steps upon $\overline{X}$; $\overline{X}_{\nu+1} = A\overline{X}_\nu$ $(\nu=1,\cdots,k-1)$

d) $\overline{X}_k = U \cdot R$ (Gram-Schmidt orthogonalisation)

e) $Q = R \cdot R^T$ (Q は $p \times p$ の小行列)

f) $Q \rightarrow B^T Q B = diag(d_1^2, d_2^2, \cdots, d_p^2)$ ($p \times p$ マトリックスの固有値問題)

g) new $X_h = U \cdot B$

h) if $(d_1/d_p)^k < 10$ then $k = k+1$

i) test for termination

原理的にはこのようであるが，実際には c) の所の一次変換の繰返しは chebyschev 加速を行う。それに伴って h) の所も何かめんどうな式を使っているが，バーチアルにとってどうこうと云う観点からはどうでも良い。また f) と g) の間に x_p の代りに random なベクトルを採用すると云う手法をそう入するがこれも今の我々の目的からはどうでも良いことである。さて A が帯で，しかもメインメモリに入りきらないほど大きいとき，c) の一次変換 k 回は，一回当り帯の端から端まで参照するから痛い。こう云うときには k をあまり急速に増さぬ方が良さそうである。p が可なり大きくて，どうしても k を増し

たいなら，　$X = AX$ の反復をAを二つに分けることにしてやると云うような工夫を要する。Aが帯だとそれが可能である。(図3－5)

chebyschev 加速のためには $\bar{x}_j$ の計算のために $\bar{x}_{j-1}$ と $\bar{x}_{j-2}$ を使うからワークエリアが余計要る。従ってそれらを含めて全部がメインメモリに入れば問題ないが，この余計のワークエリアをとったためメインメモリをはみ出すような場合には，リアルの機械だとそこで計算不能になるからすぐ判るが，バーチャルだとchebyschevをやったばかりにかえって遅くなって損をしているのに気付かない。と云うような喜劇を起すことがあり得よう。

次にKも帯，Mも帯のとき，$Kx = \lambda Mx$ の小さい方からp箇の固有値と固有ベクトルを求めることが有限要素法の動的解析などで応用上重要である。これをまともにKをコレスキー分解 LL^T して　$L^{-1}ML^{-T}\tilde{x} = (\frac{1}{\lambda})\tilde{x}$　($\tilde{x} = L^T x$)　を解く問題にしたのでは，$L^{-1}ML^{-T}$ は密行列になってしまって解けなくなる。帯の性質を保存したまゝで是非やり度いわけであるが，その有力な候補が文献(4)にあるので調べてみた。(この文献は東大生研の山田嘉昭教授に教わった。)その大筋は以下の如くであり，バーチャルにとっても有望である。

a) 初期ベクトル $x_1^{(1)}, x_2^{(1)}, \cdots x_p^{(1)}$ （正規直交）をえらぶ
それによって縦長のマトリックス $X_1 = (x_1^{(1)}, x_2^{(1)}, \cdots, x_p)$
を作る。

b) SQRT free のコレスキー分解 $K = LDL^T$ を作って置く。

c) $K\bar{X}_{k+1} = MX_k$ を $\bar{X}_{k+1}$ について解く。（$K = LDL^T$
を使って解くから速い。）

d) $K_{k+1} = \bar{X}_{k+1}^T K \bar{X}_{k+1}$ （K を $\mathcal{E}_{k+1}$ に射影。K_{k+1} は p次元）

$M_{k+1} = \bar{X}_{k+1}^T M \bar{X}_{k+1}$ （M を $\mathcal{E}_{k+1}$ に射影。M_{k+1} は p次元）

e) p次元固有値問題 $K_{k+1}Q = \lambda M_{k+1}Q$ を解く。この
解ベクトルによって作られるマトリックスを Q_{k+1} とする。

f) $X_{k+1} = \bar{X}_{k+1}Q_{k+1}$ によって新しい X_{k+1} を作る。これで
一巡した。

これで大筋は判ったが，実際実用的なプログラムとなると必要な固有値の個数をどうしてきめるかとか，その個数 r に対して Subspace の次元 p（$> r$）をどうしてきめるかとか，収束判定をどうするかと云った尾ひれがつくわけだが，バーケアルにとってこの方式はどうかの見当をつけるのには上の大筋だけチェックすれば良かろう。主要なループ c)～f) の間でできるだけドラムをまわす回数を減らすようなプログラムが作れるかどうかと云うことである。K，M の次元数を n

帯巾を $2m+1$ とするとき，$n \gg m \gg p$（例えば $n=5{,}000$，$m=200$，$p=20$）のような場合が応用上重要である。それを念頭に置くと C)，d) のステップの次の計算が主要部になる。

$X \rightarrow Z$ （$Z = MX$）

$Z \rightarrow Y$ （$LY = Z$）

$Y \rightarrow \bar{X}$ （$L^T \bar{X} = D^{-1} Y$）

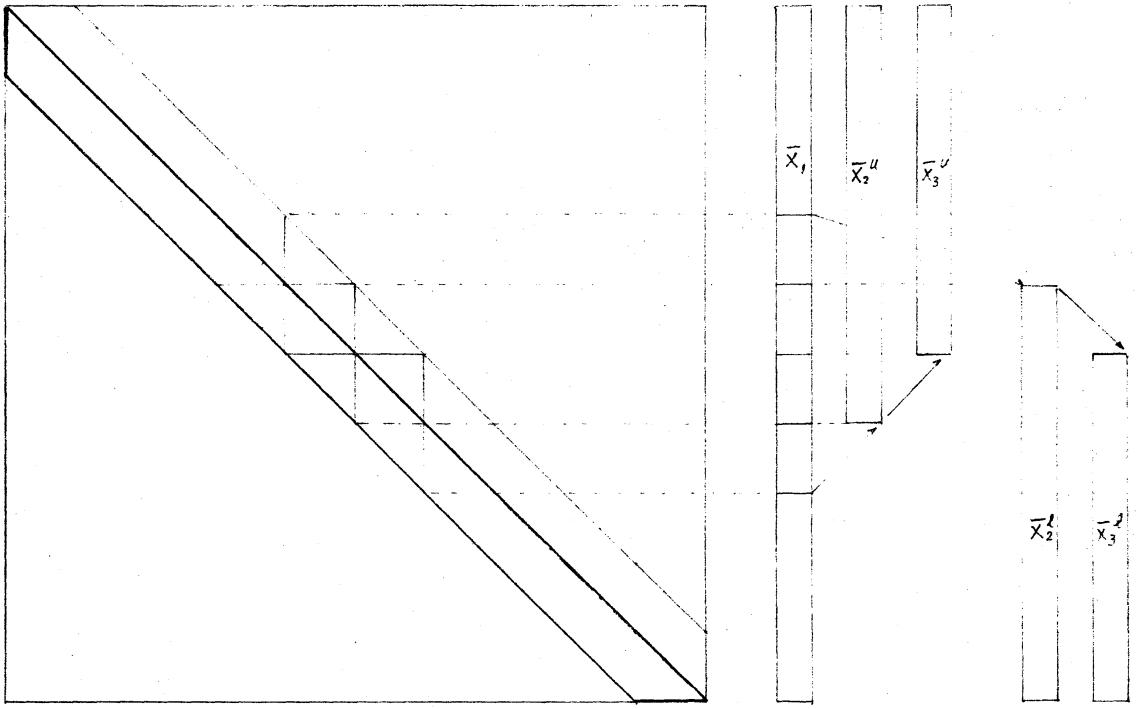
$\bar{X} \rightarrow M\bar{X}$

この間で，できるだけドラムの参照回数を減すようなアレイの設計とアルゴリズムを考えれば良い。M，L の大きさとメインメモリの容量との兼ね合いで，最適手法にも差異があらわれるが，例えば，L と M 同時に全部（と若干のワークエリア）はメインメモリに入らぬが，その半分づつなら余裕をもって入ると云うようなときには，図3－6で連想されるように，L，M をそれぞれ半分と必要な余分をよびこんで $X \rightarrow Z \rightarrow Y \rightarrow \bar{X} \rightarrow M\bar{X}$ の道すぢを約上半分と約下半分に分けてやると良い。（重複する所があることに注意）こうすれば局所性が上るだろう。プログラムするとき，対称性と帯の性質を積極的に利用することは云うまでもない。

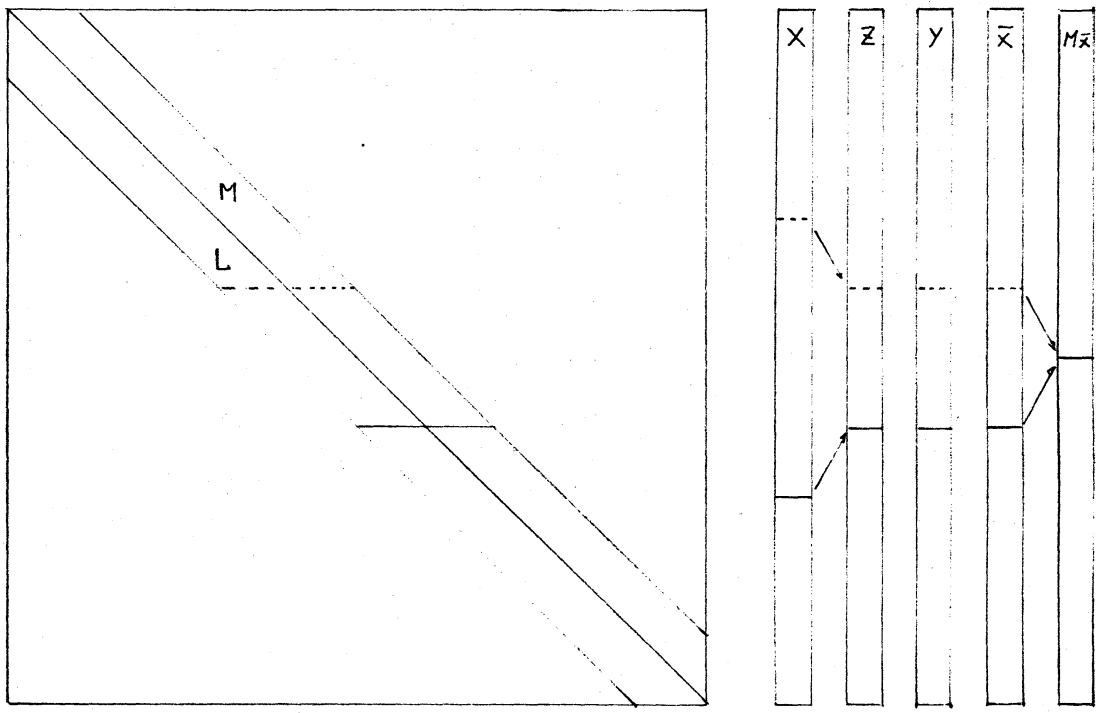
尚ほ，このアルゴリズムも，実用のプログラムとなると，用途に応じたいろいろの尾ひれがつくだろう。そう云う実用的なことは，有限要素法などを利用する方面の専門誌によく出るようである。例えば，文献(4)もその一つだし，Computers and Structures がそうである。その Vol 3 (1973) には，Corr と Jennings が Implementation of Simul. Iter. for Vibration Analysis と題して書いている。

（図 3 - 5）

$\bar{X}_{\nu+1} = A X_\nu \quad (\nu = 1, 2)$ を上下二つに分けてやる法

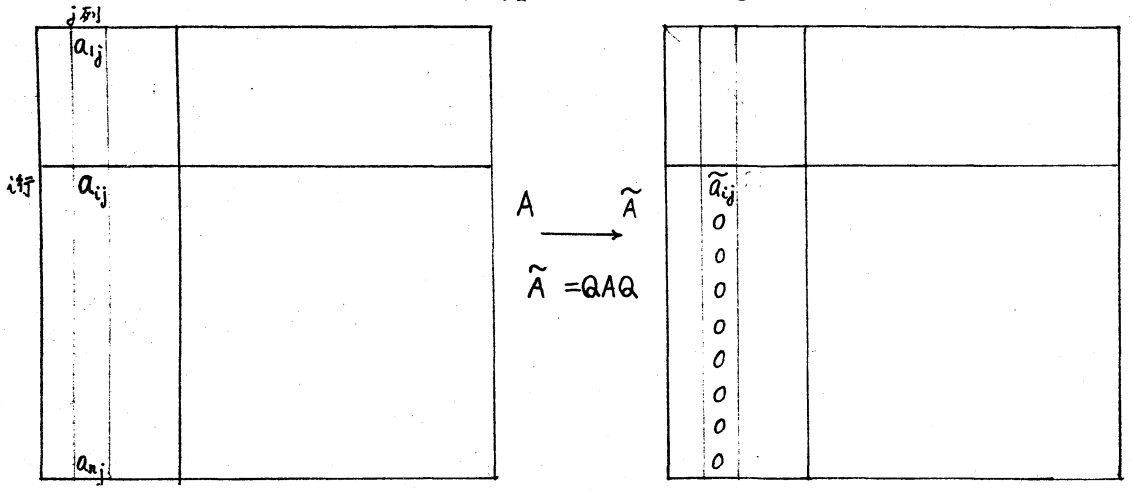


（図 3 - 6）



§4 対称帯行列に対する拡張ハウスホルダ変換

これは二種類の鏡像変換を交互に使って超大次元の対称帯行列を三重対角化,または帯に対する直接 QR 法で收束が現実的に可能な時間内で行なわれる程度にまで帯巾を縮小しようと云う目的のために筆者が研究中のもので，まだ，如何なる場合にどれ位威力があるか実際に確かめる所まで行って居ないが，局所性を良くすると云うのは具体的にどう云うことかを示す一例として，その概略を述べて見る。



上図で示されるような鏡像変換

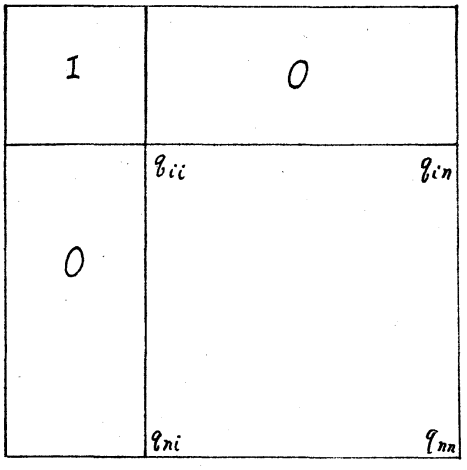
Q は式で書けば，$Q = I - \omega\omega^T/H$

但し $\omega^T = (\overbrace{0,0,\cdots,0}^{i=1\ldots r}, a_{ij}\pm S, a_{i+1,j}, \cdots, a_{nj})$

$$S^2 = a_{ij}^2 + a_{i+1,j}^2 + \cdots + a_{nj}^2$$

$$H = S^2 + |a_{ij}|\cdot S$$

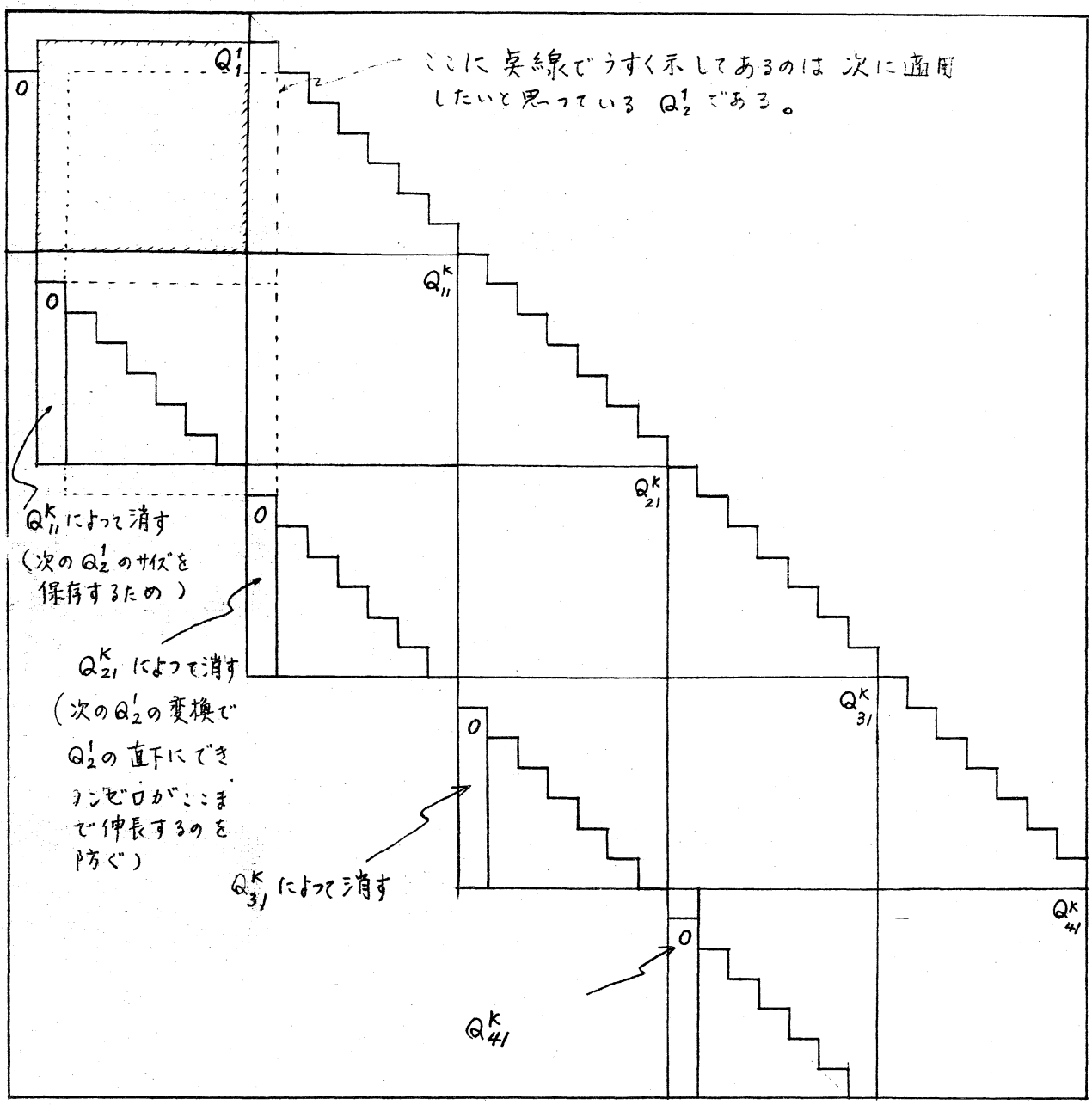
ここに $i=j+1$ にとったものを $j=1$, 2, … に対して順次使用するのが通



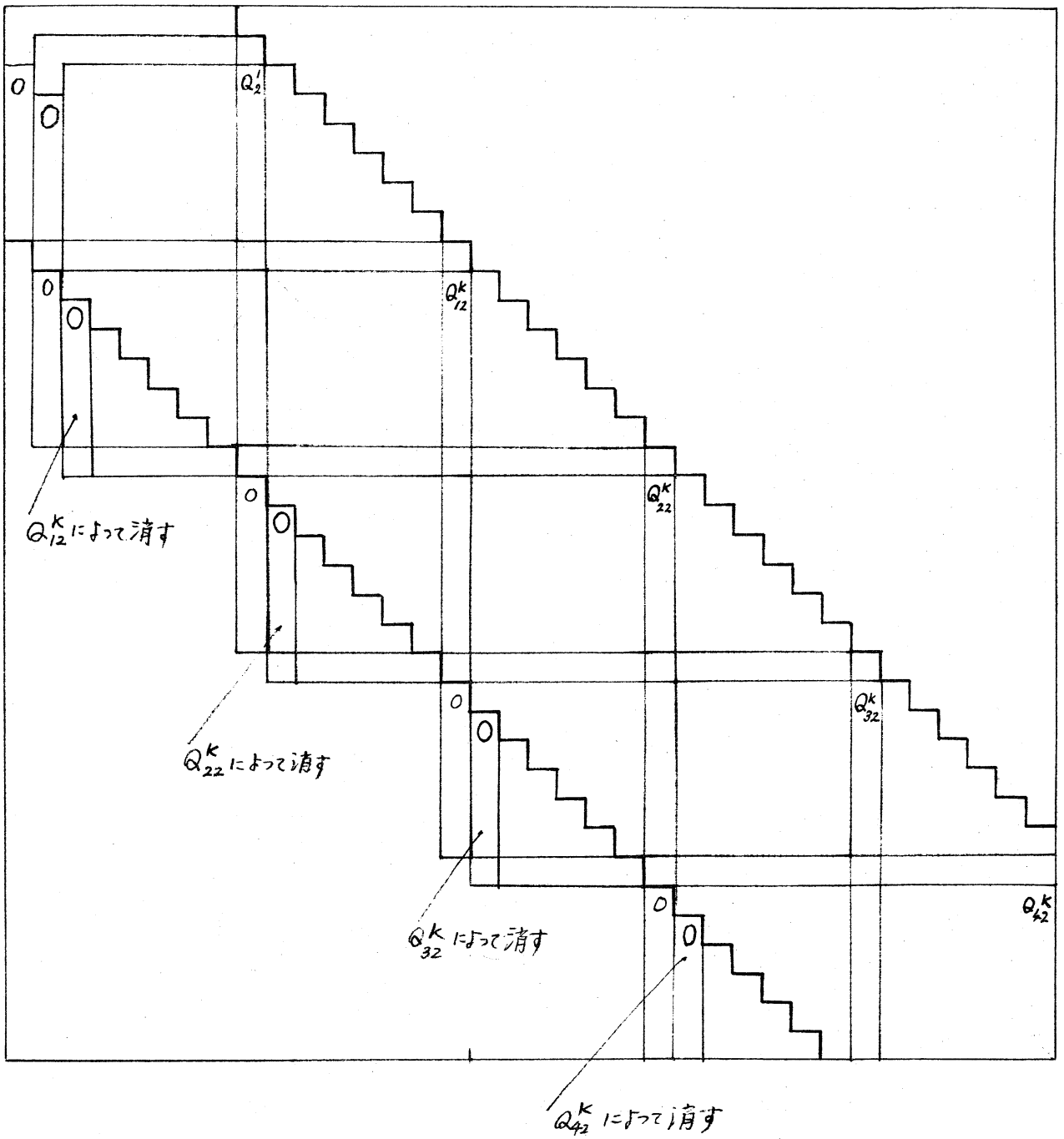
常のハウスホルダ変換である。$i=j+k$ $(k>1)$ の変換そのものは例えば文献(5)の48頁に書いてあるので周知であると思われるが，筆者のアイデアは，$i=j+1$ の変換と $i=j+k$ の k を適当にえらんだものの二種をてきとうな順序で交互に行うことによって帯巾 $2m+1$ の対称帯行列を，帯巾が途中でも高々2倍に拡がるだけで三重対角化してやろうと云うものである。式で書くのはめんどうだから図で示すことにする。もともと図を書きながら考え出したものだから。$i=j+1$ による変換を Q^1 で表し，$i=k+j$ による変換を Q^k で表すと，図4－1は，最初 Q^1_1 で第一列を三重対角化したら，そのとき Q^1 の直下にはみ出す nonzero のうち第二列だけを直ぐ Q^k_{11} で消す。そうすると Q^k_{11} の直下にまた nonzero がはみ出すがその最初の列だけ直ぐ Q^k_{21} によって消す……。こうして置いて今度は図4－2に示すように，Q^1_2 で第二列を三重対角化する。このとき Q^1_2 の直下にまた新たな nonzero がはみ出すのを Q^k_{12} で第三列だけ消す，次いで Q^k_{22} によって……。と云うわけである。

所がこの方法は判り易いけれども，一列分三重対角化する度に帯の右下端まで参照するから局所性が悪い。(勿論前の II/8 の Rutischauser-Schwarz の三重対角化よりも局所性は良いが。) そこで，図4－3　図4－4　のように改良する。

（図4-1）

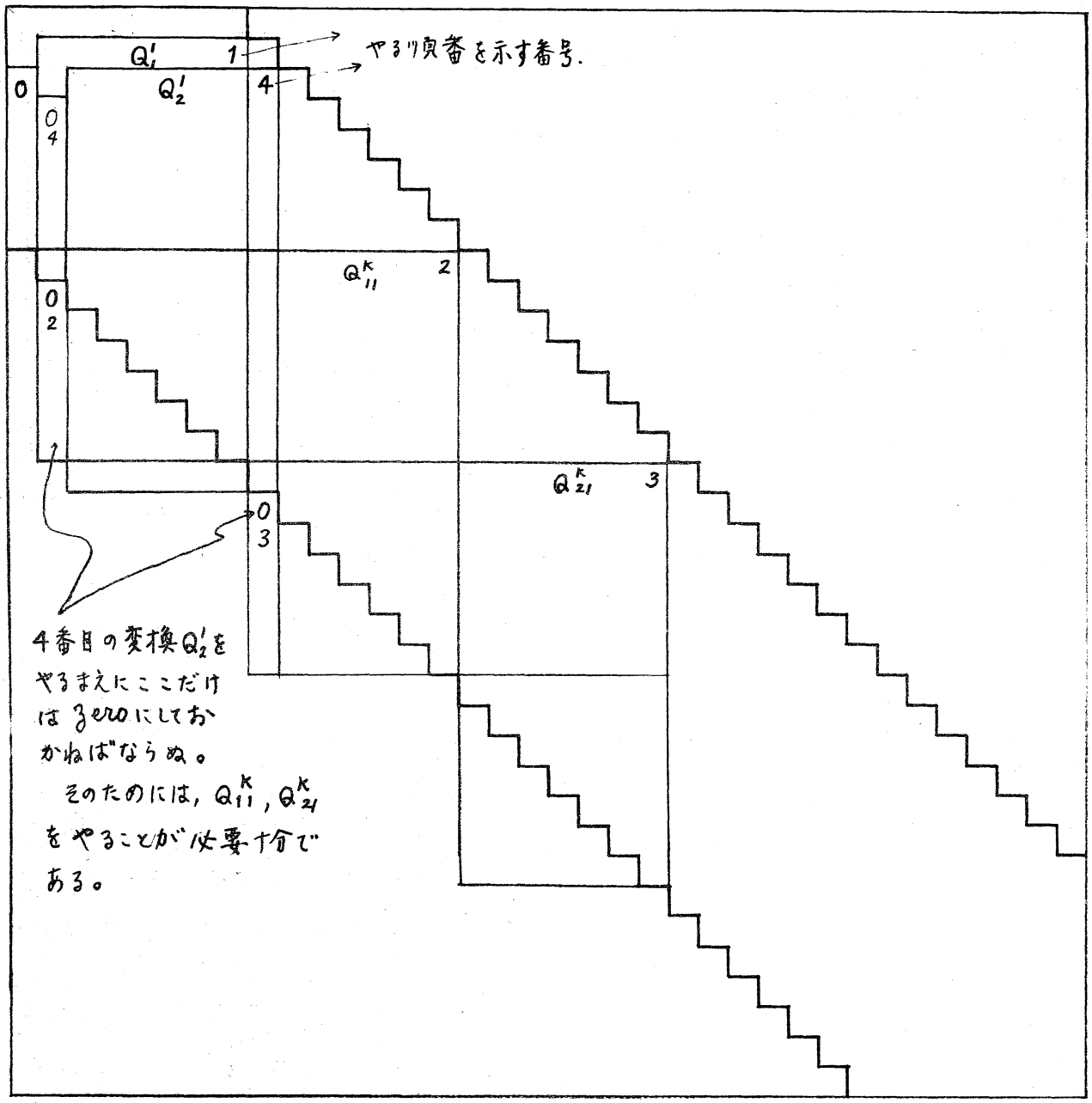

Q_1^1
0
ここに実線でうすく示してあるのは次に適用したいと思っている Q_2^1 である。
Q_{11}^K
0
Q_{21}^K
Q_{11}^Kによって消す
（次のQ_2^1のサイズを保存するため）
0
Q_{21}^Kによって消す
（次のQ_2^1の変換でQ_2^1の直下にできるゼロがここまで伸長するのを防ぐ）
Q_{31}^K
0
Q_{31}^Kによって消す
Q_{41}^K
0
Q_{41}^K

（図4-2）

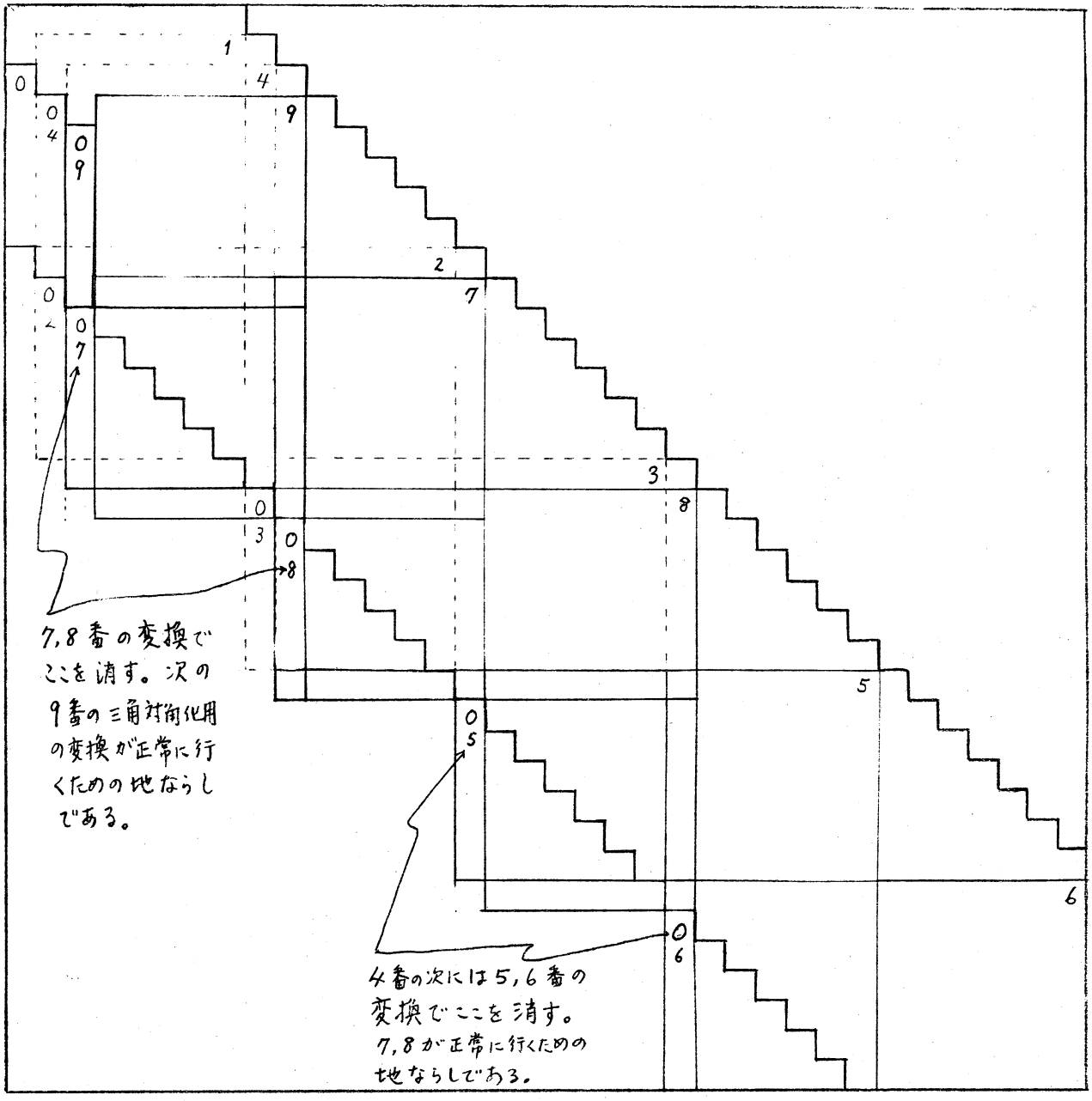

0
0
Q_2'
0
0
Q_{12}^k
Q_{12}^kによって消す
0
0
Q_{22}^k
Q_{22}^kによって消す
0
0
Q_{32}^k
Q_{32}^kによって消す
0
0
Q_{42}^k
Q_{42}^kによって消す

78

（図4－3）

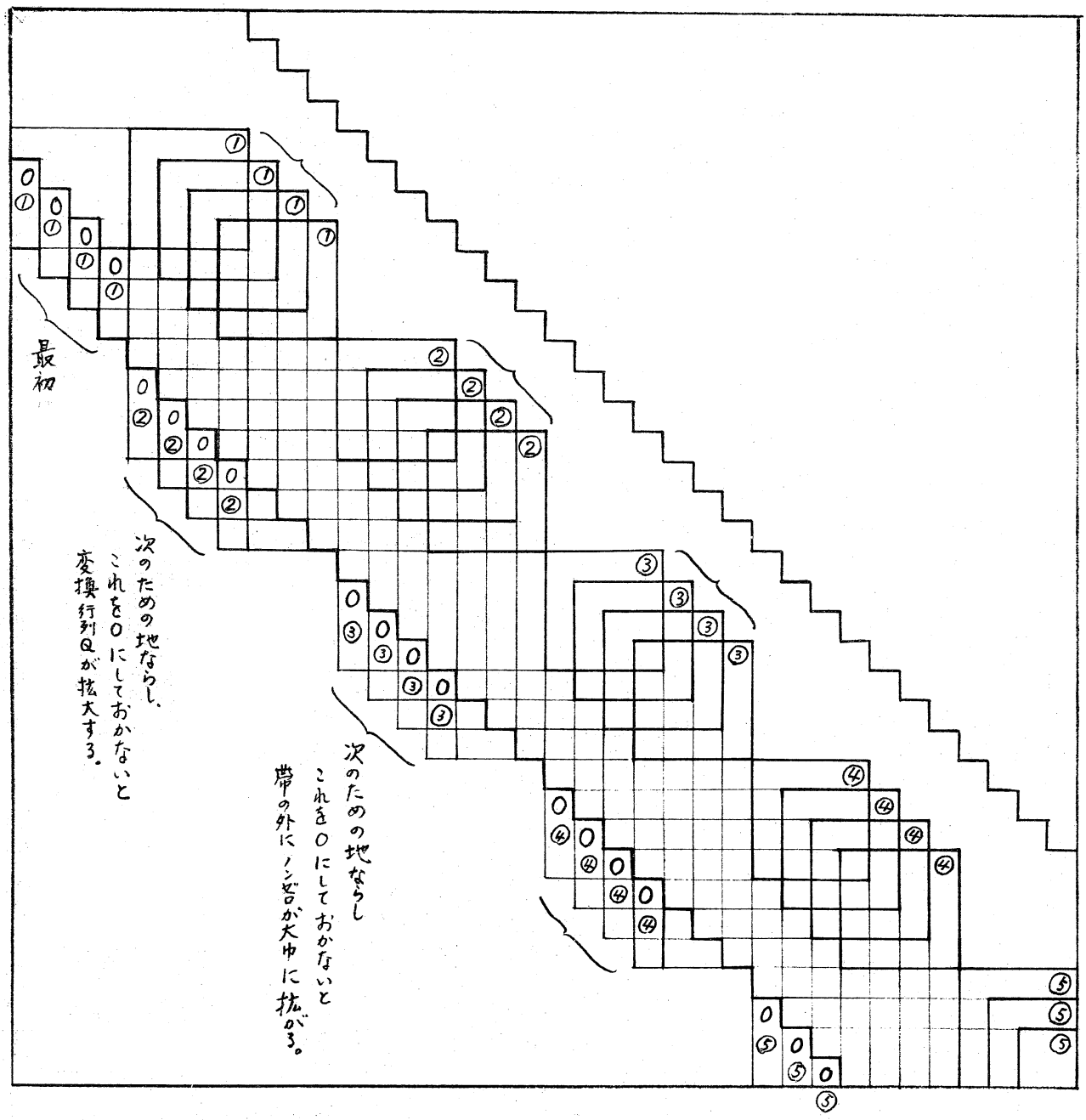

Q_1'
1
やる順番を示す番号.
0
Q_2'
4
0
4
Q_{11}^k
2
0
2
Q_{21}^k
3
0
3
4番目の変換Q_2'をやるまえにここだけはzeroにしておかねばならぬ。
そのためには，Q_{11}^k，Q_{21}^kをやることが必要十分である。

（図４－４）

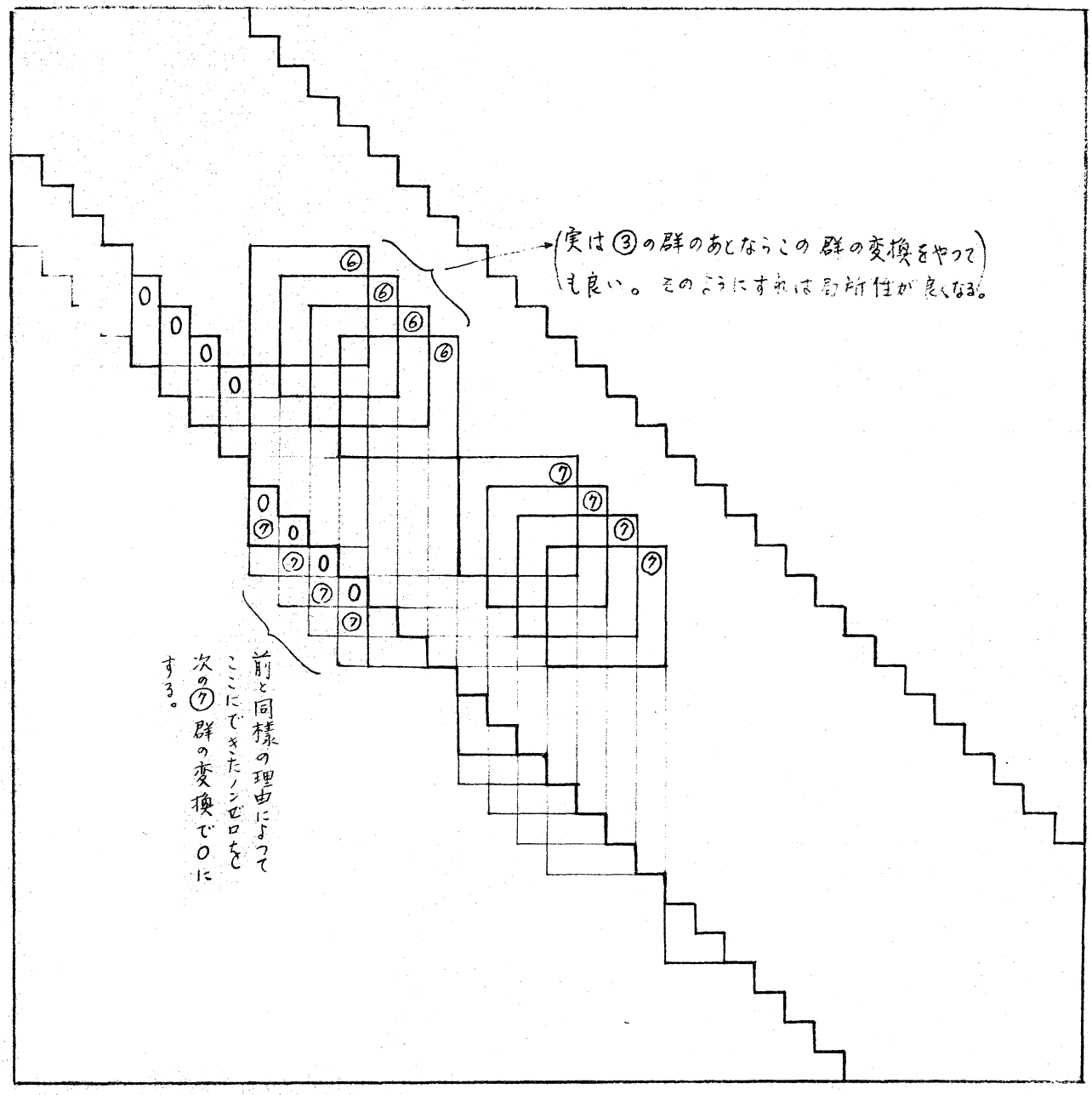

7,8番の変換でここを消す。次の9番の三角対角化用の変換が正常に行くための地ならしである。
4番の次には5,6番の変換でここを消す。7,8が正常に行くための地ならしである。

80

（図4－5）


最初
次のための地ならし、
これを0にしておかないと
変換行列Qが拡大する。
次のための地ならし
これを0にしておかないと
帯の外にノンゼロが大巾に拡がる。

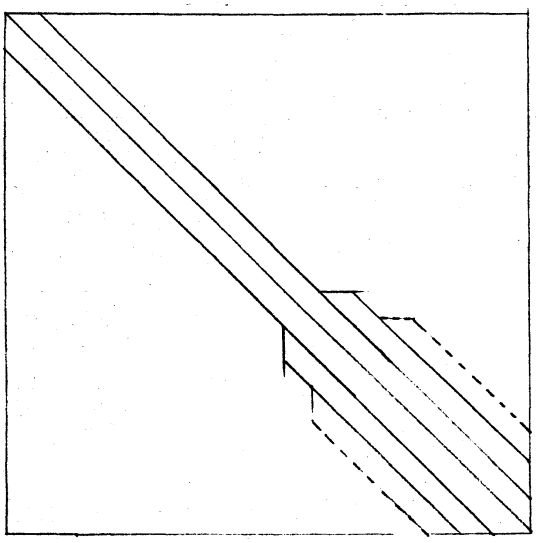
（図 4 - 6）


(実は③の群のあとならこの群の変換をやっても良い。そのようにすれば局所性が良くなる。)
前と同様の理由によってここにできたノンゼロを次の⑦群の変換で0にする。

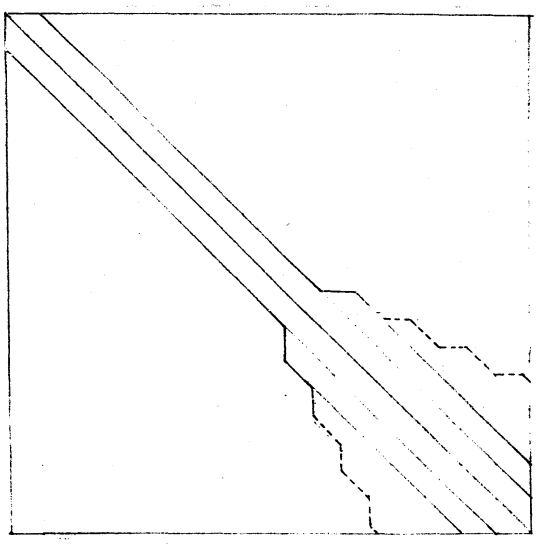
この改良の要点は，三重対角化のための変換Q_1'，Q_2'，が一定の大きさで進行するのに必要最小限の地ならしのために変換Q^kを先行させる。と云うことである。

図４－５，４－６はいきなり三重対角化と云うのでなくて帯巾を半減とか1/4とか適当な帯巾に縮小するやり方を示している。これらは，三重対角化の図４－１，４－２に対応するものであるが，図４－３，４－４に相当する改良法もあるがあまり複雑になるので書かなかった。図４－５，４－６で示されるやり方は，高層建造物の場合のように左上から右下に行くある途中から帯巾が増すような多段帯行列を，まず狭い方の帯巾にそろえると云うような実用性がある。広巾の部分の長さが短かくて，帯巾がひろがっても大したことないと云うこともあろう。図４－７が前者，図４－８が後者の場合である。

（図４－７）



（図４－８）



§5. 補遺とまとめと提言

§5-1. ランチョス法

Linear Algebra[1]にのっていないので今までとりあげなかったが，ランチョス法も超大次元の帯行列に対してならば有望と思う。(文献(6)) 勿論密行列に対しては，ハウスホルダー法の$\frac{2}{3}n^3$回の主要計算に対して，この方法は用直交化を省いても，n^3回の主要計算を要するから全く太刀打ちできないであろうが，帯に対してはハウスホルダー法が直ぐ密行列に変えてしまうのに対し，こちらは用直交化なしならば，局所性が良好だから見込みがある。問題は用直交化だが，これをWilkinsonの本(2)にあるように毎回きちんと今まで作った凡ての直交ヴェクトル系に対して新しいヴェクトルを用直交化せねばならないとなると，局所性は極度に悪くなる。だから，ランチョス法を生かす要点は局所性を保有しながら実質的に安定化に資するような用直交化法を考えることである。 今 A を与えられた対称帯行列としこれが直交変換 C によって三重対角行列 B に変換されるとして，変換 C と B の要素を決定する基本プロセスを復習すると，

$$C = (C_1, C_2, \cdots, C_i, \cdots C_n) \qquad B = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \ddots & & \\ & & & \beta_{i-1} & \alpha_i & \beta_i \\ & & & & \ddots & \ddots \end{bmatrix}$$

に対して；

$$A C_1 = \alpha C_1 + \beta_1 C_2$$

$$A C_i = \beta_{i-1} C_{i-1} + \alpha_i C_i + \beta_i C_{i+1} \qquad (A)$$

これと，$c_i^T c_{i-1} = 0$，$c_i^T c_{i+1} = 0$ を同時に満足す可きであるとの條件から，まず

$$\alpha_i = c_i^T A c_i$$

これでもし今まで $c_i^T c_{i-1} = 0$ が成立つように c_i が決められてきて居れば，$c_i^T c_{i+1} = 0$ も満足させられるように，$\beta_i c_{i+1}$ は

(A)式から決められる；　$\beta_i c_{i+1} = A c_i - (\beta_{i-1} c_{i-1} + \alpha_i c_i)$

そこで β_i と c_{i+1} を，

$$\begin{cases} c_{i+1} = A c_i - \beta_{i-1} c_{i-1} - \alpha_i c_i / \| A c_i - \beta_{i-1} c_{i-1} - \alpha_i c_i \|_2 \\ \beta_i = \| A c_i - \beta_{i-1} c_{i-1} - \alpha_i c_i \|_2 \end{cases}$$

と決めればこれで次に移れるわけである。

ここで　$v_i = A c_i - \beta_{i-1} c_{i-1} - \alpha_i c_i$ の計算で等しいに近いもの同志の引き算が起ると，桁落ちを生じて，結果として計算された v_i は　c_{i-1}，c_i と直交す可きであるにもかゝわらず直交しないものになる。また理論的には　$c_{i-2}, c_{i-3} \cdots$ とも直交して居る筈であるにもかゝわらず，これらとも直交しないものが得られてしまうことがある。そこで，これをそのまゝ放置して計算を先にすゝめると不安定を起すので再直交化が必要になる。Wilkinson はその著書(2)で，得られた v_i と凡ての $c_i, c_{i-1}, \cdots, c_2, c_1$ との内積をとつて，それらに対する平行成分 $(c_\lambda^T v_i) c_\lambda$ を凡て v_i から引き去る；　$\bar{v}_i = v_i - \sum_{\lambda=1}^{i} (c_\lambda^T v_i) c_\lambda$　ことによって再直

交化を行う可きであるとしているが，その通りやると，今まで作って来た凡てのベクトル $c_1, c_2, \cdots, c_i$ を全部参照せねばならないから，ここで計算の局所性が全く失なわれてしまう。そこで次のようなやり方はどうか？

(1) ある小さな値 eps1 (例えば 1.0E0-14) と eps2 (例えば 1.0E0-10) を予じめ決めて置く。また適当な整数 m を決めておく。

(2) v_i に混入したかもしれない c_i と c_{i-1} の平行成分；即ち $c_i^T v_i$ と $c_{i-1}^T v_i$ を計算しそれぞれ ε_i, ε_{i-1} と書く。

新しい $v_i = v_i - (c_i^T v_i) c_i - (c_{i-1}^T v_i) c_{i-1}$

とする。即ち c_i, c_{i-1} に対してだけ再直交化する。

(3) $|\varepsilon_i| + |\varepsilon_{i-1}| < eps1$ ならこの新しい v_i は立派なものだと思ってこの v_i から $c_{i+1} = v_i / \|v_i\|_2$ を作ってiを1進める。(これで今のステップは完了)

(4) $|\varepsilon_i| + |\varepsilon_{i-1}| > eps1$ なら c_{i-2} と c_{i-3} とも再直交化する；即ち $\varepsilon_{i-2} = (c_{i-2}^T v_i)$, $\varepsilon_{i-3} = (c_{i-3}^T v_i)$

新しい $v_i = v_i - (c_{i-2}^T v_i) c_{i-2} - (c_{i-3}^T v_i) c_{i-3}$

(5) $|\varepsilon_{i-2}| + |\varepsilon_{i-3}| < eps1$ ならこの新しい v_i は立ち直ったと見做してこの v_i から $c_{i+1} = v_i / \|v_i\|$ を作って … 。

(6) $|\varepsilon_{i-2}| + |\varepsilon_{i-3}| > eps1$ なら $c_{i-4}, \cdots c_{i-m}$ とも再直交化する；即ち $\varepsilon_{i-k} = c_{i-k}^T v_i$ $(k = 4, 5 \cdots, m)$

新しい $v_i = v_i - \sum_{K=3}^{m} \varepsilon_{i-K} C_{i-K}$, $\varepsilon = \sum_{K=3}^{m} |\varepsilon_{i-K}|$

(ア) $\varepsilon < \alpha \cdot eps1$ ならこの新 v_i は立ち直ったとみて…。

(イ) $\varepsilon > \alpha eps1$ なら ε を $eps2$ と比較し $\varepsilon < eps2$ なら警戒のインデケータをセットし，$eps1$ を $2eps1$ と $eps2$ の小さい方に改め計算を続行，(ウ) もし $\varepsilon \geq eps2$ なら計算をあきらめる。 ((ア),(イ)の所で使う α は $\sqrt{m}$ に近い整数をとるが良い。また(ア)に入ったときも準警戒のインデケーターをセットして置くが良い。ついでに，(ア),(イ)に入ったときの λ の値と α_i, β_i の値若干を記憶して，計算が終ったときプリントアウトするが良い。と云う意味は，計算が終ったとき或いは途中までで駄めになったとき，どのあたりから怪しくなるか，などが知り度いから。)

この際に，m の値は，$C_i, C_{i-1}, \cdots, C_{i-m}$ が主メモリに入り切って尚ほ若干の余裕があることが望ましい。従って，(今はA自身が主メモリに入らぬような超大次元の場合を考えているわけだから) m はAの帯の巾 $2m+1$ の半分以下であることが望ましい。と云うことを云っていることになる。($m = \frac{m}{2}$ つまり，帯中の1/4 程だと余り勇猛にならない。またそれで両直交化の効果は相当あると思うのだが，このあたりが数値解析と実戦の経験の両方が必要な所だと思われる。)

また，Aの要素や $C_1, C_2, \cdots, C_n$ の語長(ℓ)に対して，内部演算については加減算は語長 2ℓ で行い，乗算は($\ell \times \ell \to 2\ell$) で行うことが望ましい。(桁落ちが起るのはこれでは必ずしも救えないが，誤差のない積の防止にはなる。)

§5－2 逆反復法 を超大次元の帯について行うことは計算の主体が $(A - \mu I) x_{r+1} = x_r$ $(r = 1, \cdots)$ を x_{r+1} について解く所にあるから，バーチアルにとっては好都合のプログラムを容易に用意することができる。

§5-3 まとめ

(1) 密行列で1000元程度の実対称行列の全ての固有値を，Householder - Bisection 法でバーチャルメモリ上で1時間程度のCPU Time，4時間程度のUSE Timeでやれるようになった。これは，リアルメモリに対して約5倍のバーチャルメモリ使用である。プログラムは，二次元アレイに往復法と，中一段の次元縮約法を併用すると云うごく小手先の工夫をしたもので，勿論凡てFORTRANである。I/Oステートメントも使わない。これは相当うれしい話である。

(2) 帯に対する直接QR法，Simultaneous Iteration法，Lanczos'法はバーチャル時代に於て有望だから今後研究の密度を上げる必要がある。

(3) 帯をplane Rotationによって三重対角化するRutishauser - Schwarzの方法は，そのままでは局所性がわるいのでバーチャルには不向きである。改良はできる。

(4) 帯に対して通常のハウスホルダ三重対角化を行うと帯が直ぐにひろがって密行列になるので改良が必要である。

(5) ハウスホルダ法の一つの拡張法として，二種類の鏡像変換を交互に組合せる方法を提案した。それによって帯巾を途中であまり拡がらせないで三重対角化することができる。

また，多段帯を単一帯にそろえるとか，帯に対する直接QR法の前処理として，比較的広巾の帯を狭巾の帯に縮約することに使えるかも知れない。

§5－4 提言.

パーチアル時代になると，各々の計算法の適用範囲も変るし，新くパーチアル向きの計算法もできるだろうし，リバイバルもあると思う。一つこのあたりで日本もこの方面についての系統的且つ永続的な研究開発を，大学の共同利用センタあたりを場にしてやっていただけないものか。丁度この10年前ヨーロッパで Wilkinson－Reisch のとりまとめでこの方面に多大の貢けんがなされたように。何しろ，大次元マトリックスを解く必要は最近急速に一般化しつゝある（例えば，有限要素法，多変量解析）。たゞのプログラムテクニックなら大学に強いて頼まないが，大次元化は必然的に数値解析的な知見の深化を要求する。と云うわけでかけ出しのメーカーの若者では手に負えないことが多すぎるからである。

§6. おわりに.

バーチアル方式は　たしかに便利である。そして固有値問題でみられるように上手に使えば　不可能を可能にする　威力がある。しかしそれは　今までメモリの制限のためにやろ思ってもやれなかつたこと，或いはメモリオーバーレイと本体の内部演算速度とテープ又はディスクのアクセスデイレイ等の関係を綿密に綿密に設計してやっと成功すると云うような問題が，比較的マクロな考察さえ誤らなければうまく軌道にのると云うことであって，無條件にうれしいわけではない。また良いことには必ずうらがあるのは世の常で，従来ならメモリーオーバーと云うことで簡單にはじきとばされてしまつたプログラムも，今度は一応のみこんでしまって，ときにはひどいアイドルタイムを生じながら結果は兎も角出してしまっても，本人は良いがセンタ側にしてみればそのアイドルタイムが余りにも大きいときには他ジョブで回收し切れない。これでは他のユーザーにめいわくをかけることになる。また主メモリに入り切るような小さいジョブの場合には，個々の問題にとつては，多少のバーチァルオーバヘッドがついてリアルの場合よりも遅い一方，マルチジョブストリームのセンター運營はリアルと比べると断然効率良く行なわれる面が多

い。と云う具合に　バーチアル とはまことにもの英語の意味にふさわしく味のあるシステムである。

と云うわけではあるが何分バーチアルは今始まったばかりで，現に今始めての経験が出現しつゝあるわけで，ハード屋もソフト屋も改良に大わらわの昨今である。筆者はたまたま大型プロジェクト機の評価の一環としてバーチアルの機械の使い方についての若干の勉強と経験をしたものだからこの拙文をものすることになったわけである。固有値解析の専門家でもないものが書いたのだから間違いもあろうかと恐れるが，バーチアルを上手に使うと云うことは具体的にどう云うことかは判っていただけるのではないかと思う。　バーチアルからかりそめでなく，本當の幸福を引きだすために。

<参考文献>

(1) J.H. Wilkinson・C. Reinsch , Linear Algebra (1971)

(2) J.H. Wilkinson , The algebraic eigenvalue probrem (1965)

(3) H. Rutishauser , 'Computational aspects of....' Numer. Math 13 (1969)

(4) K.J. Bath・E.L. Wilson 'Eigen value prob. in....'
Int. J. Num. Mech. Engng. vol 6 (1973)

(5) ウオルシユ 編　高須監訳　数値解析概論　第3章 48頁 (1970)

(6) H. Takahasi・M. Natori 'Eigenvalue Probrem of Large Sparse Matrices' rep. compt centre , Univ. Tokyo , 4 (1971-1972)

附記1. 拡張ハウスホルダ変換（§4）についての補足

附1-1 理解を容易にするために。

$$\begin{pmatrix} I & 0 & 0 \\ 0 & Q & 0 \\ 0 & 0 & I \end{pmatrix} \times \begin{pmatrix} A_1 & B_1^T & C \\ B_1 & A_2 & B_2^T \\ C & B_2 & A_3 \end{pmatrix} \times \begin{pmatrix} I & 0 & 0 \\ 0 & Q & 0 \\ 0 & 0 & I \end{pmatrix}$$

$$= \begin{pmatrix} A_1 & B_1^T & C \\ QB_1 & QA_2 & QB_2^T \\ C & B_2 & A_3 \end{pmatrix} \times \begin{pmatrix} I & 0 & 0 \\ 0 & Q & \\ 0 & 0 & I \end{pmatrix}$$

$$= \begin{pmatrix} A_1 & B_1^T Q & C \\ QB_1 & QA_2Q & QB_2^T \\ C & B_2 Q & A_3 \end{pmatrix}$$

元が対称なら結果も対称であることがこれから判る。またどこがどう云う変換をうけるかを理解するのにこの図は便利である。

附1-2. 拡張ハウスホルダ変換の演算回数について

図 4-1, 4-2 に於ける変換 Q_1^1, Q_2^1, …… 即ち直接三重対角化に使う変換;

$$Q^1 = 1 - \omega\omega^T/H \ , \quad Q^1 A_1 \ , \quad (Q^1 A_1) Q^1 \ , \quad B_1 Q^1 \ , \quad Q_1 B_1^T$$

のうち, Q^1, 及び $(Q^1 A_1)\cdot Q^1$ は対称性のため他のものの約半分の演算回数で済む。また $Q_1 B_1^T$ は計算しないで良い。これらの変換は皆 m^2 回の基本演算である。そこで,

Q^1 を作るのに $\frac{1}{2}m^2$, $Q^1 A_1$ に m^2, $(Q^1 A_1) Q^1$ に $\frac{1}{2}m^2$, $B_1 Q^1$ に m^2 と云う具合に概算すると合計 $3m^2$ 回の基本演算 と云うことになる。普通に云うと $3m^2$ 回の乗算と云う所であるが, 近頃の大型電子計算機では, 浮動小数点 2倍長の乗算と同じく 2倍長の加算とほとんど同じ, 更に, STORE や BRANCH も殆んど同じか場合によってはこれらの方がかえって遅い。(例えば 360/195 や HITAC 8800) こう云うことだから, 乗算回数で云々するのは実情と合わないので, ここではむしろ基本演算と云うことで, 一次変換の最内側ループ;

```
ACC = ACC + A(I,J) * B(J,K)
```

や, 消去法にあらわれる

```
A(I,J) = A(I,J) + A(I,K) * A(K,J)
```

を一つの単位にとる。これらは, 現在の高級なコンパイラーの作り出すオブジェクトで大体 10 命令位だから, 平均命令

実行時間の10倍と考えれば良い。

さて，同様の考察によって，帯巾の拡大を防ぐための変換 Q_{11}^K, Q_{21}^K, … は

Q^K を作るのに $\frac{1}{2}m^2$，$Q^K A$ に m^2，$(Q^K A)Q^K$ に $\frac{1}{2}m^2$，$Q^K B$ に m^2

BQ^K に m^2 と云う具合に概算すると合計 $+m^2$ 回の基本演算と云うことになる。

三重対角化を完了するまでに，Q_1^1 … タイプの変換は n 回，Q_{11}^K タイプの変換は凡そ $\left[\frac{n-1}{m}\right]+\left[\frac{n-2}{m}\right]+\cdots \fallingdotseq \frac{n^2}{2m}$ 回だから，結局：

三重対角化のための総基本演算回数は	$3m^2n + \underline{2mn^2}$
ついでに SQRT は	$n + \underline{n^2/2m}$

次いで，図4-5，4-6で示した帯巾 $2m+1$ を $2p+1$ に縮減するための変換について同様の考察を行う。今度は一辺が $(m-p+1)$ の正方形の変換と，$m\times(m-p+1)$ の矩形の変換から成る。これらが，凡そ，

$$p\times\left(\left[\frac{n}{m}\right]+\left[\frac{n-p}{m}\right]+\left[\frac{n-2p}{m}\right]+\cdots\right) = \frac{p^2}{m}\left\{\frac{n}{p}+\left(\frac{n}{p}-1\right)+\left(\frac{n}{p}-2\right)+\cdots\right\}$$

$$= \frac{p^2}{m}\cdot\frac{1}{2}\left(\frac{n}{p}\right)^2 = n^2/2m$$

回で完結するから結局．必要な総基本演算回数は

帯巾を $2m+1$ から $2p+1$ に縮小のために	$\{2(m-p+1)^2+2m(m-p+1)\}n^2/2m$
ついでにSQRT は	$n + \underline{n^2/2m}$

94

アンダーラインした所が主要部であるから概算用にはそこだけで良い。今，帯巾を1度 $2m+1$ から半減；$2(\frac{m}{2})+1$ にしたのち，三重対角化するときを考えると，

SQRT ； $n^2/2(m) + n^2/2(\frac{m}{2}) = 3n^2/2m$

基本演算； $\{2(m-\frac{m}{2}+1)^2 + 2m(m-\frac{m}{2}+1)\} n^2/2m + 2(\frac{n}{2})n^2$

$\fallingdotseq \frac{3}{4} m n^2 + m n^2 = \frac{7}{4} m n^2$

となる。ついでに帯巾を1度 $2m+1$ から $\frac{2}{3}$ にしたのち三重対角化するときを考えると

SQRT ； $n^2/2(m) + n^2/2(\frac{2}{3}m) = 5n^2/4m$

基本演算； $\{2(m-\frac{2}{3}m+1)^2 + 2m(m-\frac{2}{3}m+1)\} n^2/2m + 2(\frac{2m}{3})n^2$

$\fallingdotseq \frac{4}{9} m n^2 + \frac{4}{3} m n^2 = \frac{16}{9} m n^2$.

これらを，Rutishauser-Schwarz の方法や，Lanczos（部分的角直交化）と比較するために表にして見た。この際 SQRT を別建てにしたまゝでは判り難いから，SQRT = 20×基本演算と見積って $m=10$ のときと $m=100$ のときの換算化された基本演算を示してみた。（Lanczos で帯巾 $2m+1$ に対し $\frac{m}{2}$ ～ $\frac{3m}{2}$ 本のベクトルと角直交化するとした。）

	Rutishauser - schwarz	Extended House（一度に三重対角）	Extended House（帯巾半減→三重対角）	Extended House（帯巾 2/3→三重対角）	Lanczos'（部分的角直交化）
SQRT	$n^2/2$	$n^2/2m$	$3n^2/2m$	$5n^2/4m$	n
基本演算	$4mn^2$	$2mn^2$	$\frac{7}{4}mn^2$	$\frac{16}{9}mn^2$	$(2\sim2.5)mn^2$
$m=10$ のときの換算基本演算	$50n^2$	$21n^2$	$20.5n^2$	$20.5n^2$	$(20\sim25)n^2$
$m=100$ のときの換算基本演算	$410n^2$	$200n^2$	$175n^2$	$178n^2$	$(200\sim250)n^2$
Work Area	mn	$2mn$	$1.5mn$	$1.33mn$	$(1.5\sim2.5)mn$

附記2　バーケアル向きのクラウト法について.

$$l_{11}=a_{11}\quad u_{12}=a_{12}/l_{11}\quad u_{13}=a_{13}/l_{11}\quad\cdots\quad u_{1j}=a_{1j}/l_{11}$$

$$l_{21}=a_{21}\quad l_{22}=a_{22}-l_{21}u_{12}\quad u_{23}=(a_{23}-l_{21}u_{13})/l_{22}\quad\cdots\quad u_{2j}=(a_{2j}-l_{21}u_{1j})/l_{22}$$

$$\vdots\qquad\vdots\qquad\vdots$$

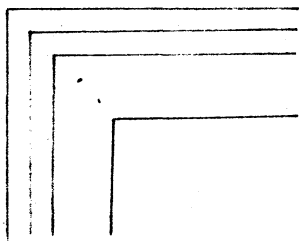
$$l_{r1}=a_{r1}\quad l_{r2}=a_{r2}-l_{r1}u_{12}\quad l_{rr}=a_{rr}-\sum_{K=1}^{r-1}l_{rK}u_{Kr}\quad\cdots\quad u_{rj}=(a_{rj}-\sum_{K=1}^{r-1}l_{rK}u_{Kj})/l_{rr}$$

$$\vdots\qquad\vdots\qquad\vdots$$

$$l_{i1}=a_{i1}\quad l_{i2}=a_{i2}-l_{i1}u_{12}\quad l_{ir}=a_{ir}-\sum_{K=1}^{r-1}l_{iK}u_{Kr}$$

$$\vdots\qquad\vdots\qquad\vdots$$

普通の教科書には上に示したように



と云う具合に左上隅から外側全部をこしらえながら次第に内側に入って行く

しかしこのままでプログラムすると上の式で＝印をしたKの動きが痛い。そこで，目的に応じて演算の順序を考えねばならぬ。まず一番簡単な場合として，$AX=B$ をたゞ一回解けは良く，しかも，A，B共に主メモリに入り切る後のことでたゞ βT_A や，τT_A をできるだけ小さくしたい位の軽い話ならば，A，Bを合併した $n\times(n+p)$ のDIMENSIONをとって，

$$\text{for } r=1,n\ \begin{cases}\text{for } \mu=1,r\ ;\ L(\mu)=a_{r\mu}-\sum_{\lambda=1}^{\mu-1}L(\lambda)u_{\lambda\mu}\\ \text{for } j=r+1,n+p\ ;\ u_{rj}=(a_{rj}-\sum_{\lambda=1}^{r-1}L(\lambda)u_{\lambda j})/L(r)\end{cases}$$

で良さそうだが，クラウト法の目的からして，

(1) 内積型の計算形式は保持する．

(2) *partial pivotting* は行うのがたてまえである．

と云うことだのにたいして，この方法は *partial pivotting* がうまく行かない。そこで止むを得ず

$$\text{for } r=1,n \begin{cases} \text{for } i=r,n \quad l_{ir} = a_{ir} - \sum_{k=1}^{r-1} l_{ik}u_{kr} & (1) \\ \text{for } \mu=1,r \quad L(\mu) = l_{r\mu} & (2) \\ \text{for } j=r+1,n+p \quad u_{rj} = \left(a_{rj} - \sum_{\lambda=1}^{r-1} L(\lambda)u_{\lambda j}\right) / L(r) & (3) \end{cases}$$

を基にして (1) と (2) の間で，$|l_{ir}|$ $(i=r\sim n)$ の *max look up* と行交換を行うことによって *partial pivotting* をすると云う手法が考えられる。しかしこれでは (3) 式の内積計算が良くなっただけで，(1) 式の内積計算は相変らずまずい。で次の方法と云うことになる。

反復法 $AX_{k+1} = X_k$ のように $AX = B$ の B を変えて同じ方程式を何度も解く場合や，$B = \{b_1, b_2, \cdots, b_p\}$ の p が n に比べてそう小さくない場合には，A の LU 分解と同時に 後退代入 ; $LY = B$, $UX = Y$ の方も重要である。これらをバーチャルオーバーヘッドすくなく解くためには，$l_{r1}, l_{r2}, \cdots l_{rr}$; $u_{rr+1}, u_{rr+2}, \cdots u_{rn}$ が それぞれこの順に並んでメモリに収容されていて欲しい。

それには，元々の a_{IJ} をあらかじめ $AT(J,I) = a_{IJ}$ の形で転置した形で収容し。出来て来る l_{IJ}, u_{IJ} は $AT(J,I)$ のう えに over write して行くと云う方針が一番良さそうである。

原理的には

for $r=1,n$

$$\begin{cases} \text{for } \mu = r,n \quad l_{\mu r} = a_{\mu r} - \sum_{K=1}^{r-1} l_{rK} u_{Kr} \\ \text{for } \lambda = 1,r \quad u_{\lambda r+1} = \left(a_{\lambda r+1} - \sum_{K=1}^{\lambda-1} l_{\lambda K} u_{K r+1} \right) / l_{\lambda\lambda} \end{cases}$$

これを実際には，凡そ次のようにやる。

for $r = 1, n$

$$\begin{cases} \text{for } \mu = r,n \begin{cases} AT(r,\mu) = AT(r,\mu) - \sum_{K=1}^{r-1} AT(K,r)\, U(K) \\ \text{search } r' \text{ such as } |AT(r,r')| \geq |AT(r,\mu)| \end{cases} \\ R(r) = r', \quad \text{for } K = 1,n \begin{cases} C(K) = AT(K,r) \\ D(K) = AT(K,r') \\ AT(K,r) = D(K) \\ AT(K,r') = C(K) \end{cases} \\ \text{for } \lambda = 1,r \begin{cases} AT(r+1,\lambda) = \left\{ AT(r+1,\lambda) - \sum_{K=1}^{\lambda-1} AT(K,\lambda)\, U(K) \right\} / AT(\lambda\lambda) \\ U(\lambda) = AT(r+1,\lambda) \end{cases} \end{cases}$$

（部分軸選択のための部分）

内積形の計算を行う DO ループの中を短かくまわすことが肝要だから，行交換，AT(I,J) で云うと列交換は本当にやった方が良い。このプログラムで，A だけが主メモリに入る位なら相当具合良く行く。A だけで主メモリをはみ出すようだともう一段の工夫が要る。